INVENTION TITLE

System and Method to distribute reasoning and pattern matching in forward and backward chaining rule engines

DESCRIPTION

Background of the Invention

[Para 1] The RETE algorithm was invented by Dr. Charles Forgy in 1978 and re-published in 1982 titled "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", in Artificial Intelligence, 19 (1982) pp. 17-37. RETE was originally designed to improve pattern matching performance. The algorithm can be divided into two parts: the rule compiler and the runtime. The rule compiler analyzes a set of rules and creates a rooted acyclic graph. The conditions of the rule are converted to one and two input nodes. The action of the rule is converted to a specialized 1-input node called terminal node. The second part of the algorithm is the runtime, which consists of a working memory and an agenda. Data requiring evaluation is added to the working memory and traverses the acyclic graph generated by the rule compiler. When all conditions of the rule are satisfied, the data triggering the rule are added to the agenda. RETE algorithm uses match-resolve-act cycles to evaluate the data. When all the rules have been fired, the evaluation process is complete. Each one input node in the acyclic graph remembers which data satisfies the condition in an alpha memory. Each two input node contains an alpha memory for the right side and a beta memory for the left side. The two input node uses the memory to remember which data matches the other side. ~~Rule engine technology is the product of research in Artificial Intelligence and intelligent systems. Throughout the 1970's, 80's and 90's, researchers have solved some complex computing challenges. One of the most efficient and well tested algorithms is RETE. It was originally described by Charles Forgy.~~

[Para 2] RETE uses the one input nodes to evaluate literal constraints, and two input nodes to evaluate joins between two data elements. An example of a literal constraint might be "the state is MA". An example of a join might be "the zip code of the billing address is equal to the zip code of the mailing address." The rule compiler analyzes the rules for similarities

and organizes the nodes to optimize performance and scalability. Forward chaining rule engines are said to be data driven. The rule engine can only reason over data that is added to the working memory.

Summary of the Invention

[Para 3] The invention solves performance issues previously ignored by researchers in the field of artificial intelligence. Using existing techniques, a rule engine implementing Dr. Forgy's original RETE algorithm cannot reason over extremely large datasets or partitioned data. The invention solves the following limitations.

- Dividing the pattern matching between multiple rule engines and share the partial and complete matches efficiently.
- Efficiently dividing the pattern matching across several rule engines dynamically.
- Efficiently distributing the indexes of the working memory across multiple machines.
- Efficiently distributing one or more conditional elements of a rule to remote systems and dividing the work across a cluster of computer systems.

[Para 4] In general, one form of the invention is a computer system which implements the RETE extension to perform distributed reasoning. The system distributes reasoning by replicating the memory indexes across the systems. This reduces the memory requirements of each system and reduces the quantity of data each rule engine sends to the others. Using this approach, each rule engine in the cluster has a subset of the entire dataset. A system implementing the distributed RETE extension can be used in a variety of applications like order management, regulatory compliance, military command control applications or business process automation. The invention is particularly well suited to large real-time systems like military command control systems or security trading systems, which partition data across several locations.

Detailed Description of the preferred embodiment

[Para 5] RETE defines a method for converting if/then/else statements to executable code. The if/then/else statements represent business rules that define business decisions. For example, an user may define a rule for managing orders "If the items of the order are in stock and the credit card is valid, then send the order to fulfillment". In the example, "if the items of the order are in stock and the credit card is valid" are the conditions of the rule. The action of the rule is "send the order to fulfillment." The rule engine translates the conditions of the rule to a graph also known as a directed acyclic graph. During the translation process, it identifies similarities between rules and optimizes the graph to improve performance. In the situation were multiple rules share the same condition, the rule engine will determine if the nodes in the graph can be shared between rules. This reduces the number of evaluations a rule engine has to perform on the data. An example RETE graph is provided in Figure 1. RETE is capable of evaluating tens of thousands or hundreds of thousands of rule efficiently and only evaluates the rules that apply to the data. Prior to the invention of RETE, many systems had to evaluate all the rules for every piece of data. This means as the rule count or dataset grows, it takes longer to evaluate. A system using RETE is able to provide better performance than naïve systems that evaluate all rules for every data element.

[Para 6] RETE has several limitations, which affect the performance and scalability. When the dataset is over 10 million records, the system may not have enough physical memory. In those situations, current systems cannot compute the result and would like fail catastrophically. Products on the market today cannot reason over extremely large datasets over 50 million records with thousands of complex business rules.

[Para 7] ~~Distributed reasoning, unlike collaborative agents provides methods by which multiple rule engines reason over large datasets in real-time. Whereas collaborative agents~~

~~use agents to reason over discrete problems, it cannot reason over large sets of data. Furthermore, collaborative agents" techniques require the rule engine to load all necessary data within the same engine. For small datasets, these techniques prove to be powerful, but they do not scale for large datasets.~~ The invention solves the memory limitation by dividing the data into partitions. Instead of having the entire dataset in a single rule engine, the data is divided across dozens or thousands of rule engines across many computer systems. There are two primary benefits of partitioning the data. First, pattern matching is performed in parallel by multiple rule engines. Second, a rule engine can reason over data in another engine thousands of miles away in another location. This is achieved by replicating the 2-input node memory indexes across systems sharing the same rules or nodes. When the engine evaluates a fact, it uses the index to perform the comparison. Any changes to the indexes are sent to the other systems periodically. This enables businesses to build a large cluster of cheap servers to reason over hundreds of millions of records in real-time.

[Para 8] Technically, a forward chaining rule engine utilizing distributed reasoning and pattern matching ~~RETE algorithm~~, can reason over large datasets when nodes are distributed to other systems. This allows the system to match on a specific ~~instance of a class (also called business object)~~ data element without requiring the object instance reside in the same working memory and be locally available. It is important to note the engine will reason over shared data resident in other engines using data steams or indexes. This reduces the memory requirements and improves efficiency. It also enables the engines to share all or part of their reasoning network and distribute the process. Distributing nodes ~~of a set of rules~~ across multiple systems allows each engine to perform pattern matching on a single ~~object instance~~ data element and route the results to the originating system. Unlike load balancing techniques, a true distributed reasoning system does not require all systems of a cluster to deploy identical sets of rules, which creates redundant rules within the environment and increases maintenance costs. In a distributed reasoning/distributed

pattern matching system, each system deploys a different set of rules (also called module or rule set), based on its own needs and configuration requirements. At runtime, the rule engines monitor resource utilization and distribute nodes dynamically and on demand. In some cases, it may be desirable to deploy the same set of rules across multiple instances. At runtime, the cluster would intelligently partition the data.

[Para 9] Prior ~~techniques~~ research with pattern matching on extremely large datasets relied on load balancing techniques to prioritize and categorize atomic processes. This approach requires every system ~~to~~ have all required data locally, leading to multiple data caches. In a production environment with large datasets, each system cannot load all required data and data synchronization becomes a ~~potential~~ problem.

[Para 10] The invention described in the patent was previously impractical due to the processing power of personal computers. Researchers in the field of Artificial intelligence did not consider data partitions with replicated indexes. The invention is currently feasible due to advances in the processing power of personal computers and high-speed networks. With this invention, it is possible to build large distributed reasoning systems using hundreds and thousands of personal computers. By distributing the indexes and facts between thousands of systems effectively, a forward and backward chaining rule engine can reason over hundreds of millions of facts using inexpensive hardware. Without this invention, rule engines cannot reason over extremely large datasets.

Description of the Drawings

[Para 11] Fig. 1 is a network diagram of an example rule with four one input, one two input and one terminal node. The example rule contains one literal constraint for each object type and one join node between the two data elements.

[Para 12] Fig. 2 is a block diagram illustrating data partitioned across several rule engines. Each engine has the same rules, but may also contain other rules, which are not shared by other engines. A set of data is divided across several engines to improve scalabilty and performance, enabling parallel processing.

[Para 13] Fig. 3 is a block diagram, which shows 2 rule engines starting with different set of rules at startup. Some time later at T_n some nodes from one rule engine are distributed to the other rule engine. Optionally, both system can start with the same set of rules. In both cases, the data is partitioned as Fig. 2 illustrates.